**五子棋**

10300240039 章超

程序设计综合作业

一、目的和要求

通过综合作业，让学生综合应用所学程序设计的知识和能力，独立完成一个相对比较完整的课题，以进一步提高学生的程序设计能力。要求每个学生从供选作业中任选一个。

二、提交内容

1、C语言源程序；

2、设计文档，详细说明问题的设计、分析过程，程序结构、各函数的功能和算法。并对程序调试、上机操作过程给出一定的描述。

3、把所有文件压缩为一个.rar文件,注明姓名、学号，项目名称（如：021002-王小明-五子棋.rar）

4、提交至校虚拟教学网站http://vcampus.fudan.edu.cn；

三、项目介绍

见上机指导实验10。

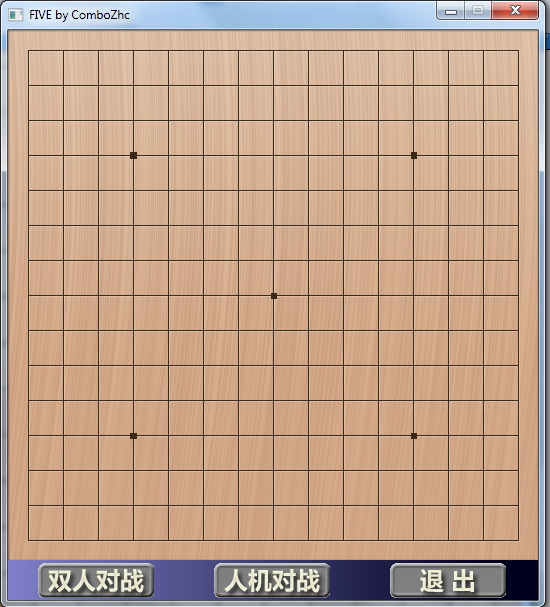
**设计**

本程序采用SDL架构来显示图形界面. 由于需要调用MFC,故必须要安装Visual Studio 2010 Redistributable.

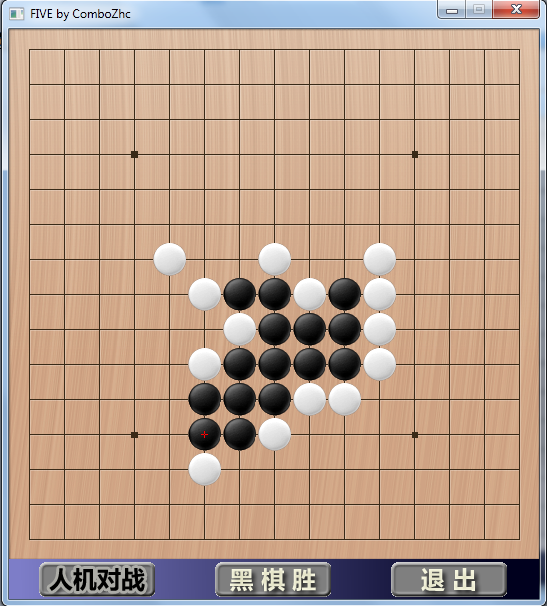
[](http://baike.baidu.com/image/906289dd1a5a31205982dd10)

SDL（Simple Directmedia Layer）被广泛的用于许多著名的游戏。最著名的游戏是赢得LIUNX组游戏开发大奖的 文明：权利的召唤（Civilization: Call To Power）。

下图为游戏界面:



初始界面



游戏界面

上部棋盘为游戏区域，能够显示上步着棋位置与当前待着棋位置。

下部按钮为控制区域，支持人机对战和双人对战两种，支持悔棋，支持人机对战的先后手选择。

**算法**

本五子棋程序采用贪心算法，引入了博弈中的极大极小化算法。

核心算法是通过枚举AI着棋位置和玩家下一步着棋位置来计算AI最优着棋位置。

(1)AI着棋价值的函数表达式为

其中,

计算.

连成子的基础分值为,

若棋子中间有空格,则分值相应减小;

若单面有对方棋子阻挡,则分值除以

若双面均有对方棋子阻挡,则分值再除以;

若对方两个阻挡棋子之前已经不能形成五子连珠,则直接赋为;

若两个以上方向都有活棋,则分值乘以;

时间即已下步数,

当时间越靠前,离玩家上一步着棋位置越近,分值越高,

(2)对于杀棋(双三、双四、四三、活四)等,

若

若;

其中,

（此种判断适用于大部分情况，对于某些通过冲四来抵挡双三的着法不适用）

AI枚举时，先判断杀棋，后判断着棋价值。这里值得一提的是，**如果出现杀棋时，本程序不采用break来缩短计算时间**，使玩家通过观察AI反应时间来寻找杀棋位置。

以上为核心算法。

参数均在自己和同学们的测试中不断调整，改进。

游戏控制操作方面，

本程序运用三个按钮指针来分离指针显示和指针切换部分。

游戏过程函数void game(int flag)运用flag标识来表明人机对战(false)与双人对战(true)。

**特性**

本五子棋程序采用SDL架构，图形化界，面鼠标操作，简单友好易用。

本五子棋程序支持人机对战和双人对战两种，支持悔棋，支持人机对战的先后手选择。

本五子棋程序能够显示上步着棋位置以及当前着棋位置，避免误操作。

本五子棋程序支持强力贪心AI， 包括双三、双四、四三的杀棋判断，但没有判断禁手。

本五子棋出现杀棋时，运算时间始终稳定，避免玩家因为AI做出快速判断而发现杀棋。

**程序调试**

本程序调试运用Visual Studio 2010的编辑环境。游戏控制部分以Visual Studio 2010 IDE直接调试。而AI部分主要以屏幕和文件输出为主，通过记录AI所下每一步棋的位置和分值来发现问题，特别考虑了对于特意让AI获胜的情况以及各类杀棋的判断。由于程序运行方便，故也请其他几位同学帮忙一同测试，若发现问题则可直接把记录文件传输于我，达到高效率的调试。

具体调试文件内容

//Record.txt

游戏模式

玩家著棋位置

电脑着棋位置

悔棋步数

//Stdout.txt

游戏模式

电脑着棋试探位置以及玩家下一步的最优应招

电脑着棋试探位置的估价分值

电脑最终选择的位置

调试除了人机对战，还能通过打开两个程序实现机器自我对战，也是测试参数设置的一种优良方法。

**源程序 注释 调试信息**

//FIVE.c

#pragma once

#pragma comment(lib,"SDL.lib")

#pragma comment(lib,"SDLmain.lib")

#pragma comment(lib,"SDL\_image.lib")

//载入静态链接库

#include "SDL/SDL.h"

#include "SDL/SDL\_image.h"

//载入SDL库

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <math.h>

#define SCREEN\_WIDTH 530

#define SCREEN\_HEIGHT 570

#define SCREEN\_BPP 32

//530 \* 570 , 32位色

#define TIME\_DELAY 10

//屏幕刷新时间

#define COLORKEY\_R 200

#define COLORKEY\_G 0

#define COLORKEY\_B 200

//扣色RGB:紫色(200,0,200)

#define BUTTON\_WIDTH 120

#define BUTTON\_HEIGHT 40

#define BUTTON1\_X (SCREEN\_WIDTH / 6 - BUTTON\_WIDTH / 2)

#define BUTTON1\_Y (SCREEN\_HEIGHT - BUTTON\_HEIGHT)

#define BUTTON2\_X (SCREEN\_WIDTH / 6 \* 3 - BUTTON\_WIDTH / 2)

#define BUTTON2\_Y (SCREEN\_HEIGHT - BUTTON\_HEIGHT)

#define BUTTON3\_X (SCREEN\_WIDTH / 6 \* 5 - BUTTON\_WIDTH / 2)

#define BUTTON3\_Y (SCREEN\_HEIGHT - BUTTON\_HEIGHT)

#define BUTTON\_STATUS 3

//按钮位置及宽和高

#define DIST\_MOVE 15

//此为落子判断, 若离棋盘中心点过远则判断为不落该子, 避免误操作

#define SQUARE\_POS 20

#define SQUARE\_WIDTH 35

#define PIECE\_WIDTH 33

//棋盘左上角位置,棋盘宽度,棋子宽度

#define TOTAL\_LINE 15

#define TOTAL\_PIECE (TOTAL\_LINE) \* (TOTAL\_LINE)

#define TOTAL\_DIRECTION 4

#define TOTAL\_STYLE 17

//总行数/总棋子数/总方向/总棋型数

#define TOTAL\_JUDGE 11

//判断用的字符串总长度

#define TWO 2

#define FIVE 5

#define TEN 10

#define INFI 100000000

// 2, 5, 10, 无穷大

#define SCORE\_DEFEND 9900

//需要防御的分值

#define SCORE\_MUST\_DEFEND 99000

//必须防御的分值

#define SCORE\_TERMINAL 1000000

//杀棋的分值

FILE \*fp;//日志文件

enum PIECE {NOTHING, BLACK, WHITE};//棋子

int systemOver = 0; //系统结束

SDL\_Surface \*pScreen;

SDL\_Surface \*pBackground;

SDL\_Surface \*pPiece;

SDL\_Surface \*pButton;

//SDL的图片读取变量

struct SDL\_Rect rPieceBlack;

struct SDL\_Rect rPieceWhite;

struct SDL\_Rect rPieceLast;

struct SDL\_Rect rPieceTarget;

//SDL\_Rect矩形, 用于放置图片到屏幕

struct SDL\_Rect rButtonRegret[BUTTON\_STATUS];

struct SDL\_Rect rButtonQuit[BUTTON\_STATUS];

struct SDL\_Rect rButtonHuman[BUTTON\_STATUS];

struct SDL\_Rect rButtonComputer[BUTTON\_STATUS];

struct SDL\_Rect rButtonBlack;

struct SDL\_Rect rButtonWhite;

struct SDL\_Rect rButtonDraw;

struct SDL\_Rect rButtonLater[BUTTON\_STATUS];

//按钮的SDL\_Rect矩形数组(悔棋，退出，双人，人机，黑棋胜，白棋胜，平局, 后手)

struct SDL\_Rect \*prButton1;

struct SDL\_Rect \*prButton2;

struct SDL\_Rect \*prButton3;

//按钮的SDL\_Rect矩形指针, 用于切换不同按钮

enum PIECE map[TOTAL\_LINE][TOTAL\_LINE];//棋盘

int stepX[TOTAL\_PIECE + 1];

int stepY[TOTAL\_PIECE + 1];//每步记录

int regretStep = -1;//悔棋位置

int moveX[TOTAL\_DIRECTION] = {1, 1, 0, -1};

int moveY[TOTAL\_DIRECTION] = {0, 1, 1, 1};//方向数组

char\* style[TOTAL\_STYLE] = {

"111111", "11111", "1111", "10111", "11011",

"11101", "111", "1011", "1101", "11001",

"10101", "10011", "11", "101", "1001",

"10001", "1"};//棋型字符串

int score[TOTAL\_STYLE] = {

1000000, 1000000, 100000, 99000, 99000,

99000, 10000, 9900, 9900, 2000,

1900, 2000, 1000, 500, 300,

200, 100};//棋型分数

int styleWin[TOTAL\_STYLE] = {

6, 5, 4, 4, 4,

4, 3, 3, 3, 2,

2, 2, 2, 2, 2,

1, 1};//棋型对应的棋子数

int styleLen[TOTAL\_STYLE];//棋型长度

//填充SDL\_RECT

//参数:r:填充的SDL\_RECT地址, (x,y,w,h):(x坐标, y坐标, 宽度, 高度)

void fillRect(struct SDL\_Rect\* r, int x, int y, int w, int h) {

r->x = x; r->y = y;

r->w = w; r->h = h;

}

//最大值

//返回:最大值

//参数:(a,b)两个待比较值

int maxTwo(int a, int b) {

return a > b ? a : b;

}

//时间估价

//返回:时间估价值

//参数:count:步数

//步数越大时间估价越趋向于1

//用于AI的攻和守之转换

double timeMul(int count) {

return exp(-1.0/count);

}

//距离估价

//返回:距离估价值

//参数:count:步数, (disX, disY):参考位置

//距离越近越大

//用于AI的着棋

double distPlus(int count, int disX, int disY) {

return 1 + TEN/(count \* count \* count)/(maxTwo(abs(stepX[count - 1] - disX),abs(stepY[count - 1] - disY)));

}

//相同分数的位置数估价

//返回:位置数估价值

//参数:位置数

//相同分数位置越多估价越高

//用于AI的防守

double mulMul(double x) {

return 1 + x/10.0;

}

//是否为出界位置

//返回:出界为1

//参数:(posX, posY):判断位置

int out(int posX, int posY) {

return (posX < 0 || posX >= TOTAL\_LINE || posY < 0 || posY >= TOTAL\_LINE);

}

//是否为鼠标事件

//返回:鼠标事件为1

//参数:SDL\_Event

int validMove(union SDL\_Event\* gameEvent) {

return gameEvent->type == SDL\_MOUSEMOTION || gameEvent->type == SDL\_MOUSEBUTTONDOWN || gameEvent->type == SDL\_MOUSEBUTTONUP;

}

//是否为鼠标在着棋位置

//返回:着棋位置为1(一维)

//参数:x:着棋位置(一维)

int validPieceMove(int x) {

return x <= SCREEN\_WIDTH && ((x - SQUARE\_POS) % SQUARE\_WIDTH <= DIST\_MOVE || (x - SQUARE\_POS) % SQUARE\_WIDTH >= SQUARE\_WIDTH - DIST\_MOVE);

}

//鼠标单击按钮

//返回:单击了哪个按钮

//参数:(x,y):鼠标位置

int getButtonMove(int x, int y) {

if (x - BUTTON1\_X <= BUTTON\_WIDTH && x - BUTTON1\_X >= 0 &&

y - BUTTON1\_Y <= BUTTON\_HEIGHT && y - BUTTON1\_Y >= 0)

return 1;

if (x - BUTTON2\_X <= BUTTON\_WIDTH && x - BUTTON2\_X >= 0 &&

y - BUTTON2\_Y <= BUTTON\_HEIGHT && y - BUTTON2\_Y >= 0)

return 2;

if (x - BUTTON3\_X <= BUTTON\_WIDTH && x - BUTTON3\_X >= 0 &&

y - BUTTON3\_Y <= BUTTON\_HEIGHT && y - BUTTON3\_Y >= 0)

return 3;

return 0;

}

//鼠标着棋

//返回:着棋的位置(一维)

//参数:x:鼠标位置(一维)

int getPieceMove(int x) {

if ((x - SQUARE\_POS) % SQUARE\_WIDTH <= DIST\_MOVE)

return (x - SQUARE\_POS) / SQUARE\_WIDTH;

else

return (x - SQUARE\_POS) / SQUARE\_WIDTH + 1;

}

//棋子转换成字符

//本方棋子'1' 对方棋子'2' 无棋子'0'

char getPieceChar(enum PIECE x, enum PIECE t) {

if (t == BLACK)

switch(x) {

case NOTHING: return '0';

case BLACK: return '1';

case WHITE: return '2';

}

else

switch(x) {

case NOTHING: return '0';

case BLACK: return '2';

case WHITE: return '1';

}

return '0';

}

//反色棋子

//棋子的黑白转换

enum PIECE getReversePiece(enum PIECE x) {

if (x == BLACK)

return WHITE;

if (x == WHITE)

return BLACK;

return NOTHING;

}

//清空地图

//清空着棋记录

void fresh() {

int i, j;

for (i = 0; i < TOTAL\_LINE; ++i)

for (j = 0; j < TOTAL\_LINE; ++j)

map[i][j] = NOTHING;

for (i = 0; i < TOTAL\_PIECE + 1; ++i)

stepX[i] = stepY[i] = -1;

}

//刷新屏幕

void flip(int count) {

int i, j;

struct SDL\_Rect rTemp;

//显示背景

SDL\_UpperBlit(pBackground, NULL, pScreen, NULL);

//显示按钮

fillRect(&rTemp, BUTTON1\_X, BUTTON1\_Y, 0, 0);

SDL\_UpperBlit(pButton, prButton1, pScreen, &rTemp);

fillRect(&rTemp, BUTTON2\_X, BUTTON2\_Y, 0, 0);

SDL\_UpperBlit(pButton, prButton2, pScreen, &rTemp);

fillRect(&rTemp, BUTTON3\_X, BUTTON3\_Y, 0, 0);

SDL\_UpperBlit(pButton, prButton3, pScreen, &rTemp);

//显示棋子

for (i = 0; i < TOTAL\_LINE; ++i)

for (j = 0; j < TOTAL\_LINE; ++j) {

fillRect(&rTemp, SQUARE\_POS + SQUARE\_WIDTH \* i - PIECE\_WIDTH / 2,

SQUARE\_POS + SQUARE\_WIDTH \* j - PIECE\_WIDTH / 2, 0, 0);

switch (map[i][j]) {

case BLACK: SDL\_UpperBlit(pPiece, &rPieceBlack, pScreen, &rTemp); break;

case WHITE: SDL\_UpperBlit(pPiece, &rPieceWhite, pScreen, &rTemp); break;

}

if (i == stepX[count + 1] && j == stepY[count + 1])

SDL\_UpperBlit(pPiece, &rPieceTarget, pScreen, &rTemp);

if (i == stepX[count]&& j == stepY[count])

SDL\_UpperBlit(pPiece, &rPieceLast, pScreen, &rTemp);

}

//翻转屏幕!

SDL\_Flip(pScreen);

}

//初始化

void init() {

int i;

//日志文件初始化

fp = fopen("record.txt", "w");

//棋型长度数组初始化

for (i = 0; i < TOTAL\_STYLE; ++i)

styleLen[i] = strlen(style[i]);

//随机初始化

srand(time(NULL));

//SDL初始化

SDL\_Init(SDL\_INIT\_EVERYTHING);

SDL\_WM\_SetCaption("FIVE by ComboZhc", 0);

//屏幕初始化

pScreen = SDL\_SetVideoMode(SCREEN\_WIDTH, SCREEN\_HEIGHT, SCREEN\_BPP, 0);

//图片初始化

pBackground = SDL\_LoadBMP("images\\Background.bmp");

pPiece = SDL\_LoadBMP("images\\Piece.bmp");

SDL\_SetColorKey(pPiece, SDL\_SRCCOLORKEY, SDL\_MapRGB(pPiece->format, COLORKEY\_R, COLORKEY\_G, COLORKEY\_B)); //扣色

pButton = SDL\_LoadBMP("images\\Button.bmp");

SDL\_SetColorKey(pButton, SDL\_SRCCOLORKEY, SDL\_MapRGB(pButton->format, COLORKEY\_R, COLORKEY\_G, COLORKEY\_B));//扣色

//SDL\_RECT初始化(记录该精灵图片在源图片中的位置，宽度和高度)

fillRect(&rPieceBlack, 0, 0, PIECE\_WIDTH, PIECE\_WIDTH);

fillRect(&rPieceWhite, 0, PIECE\_WIDTH, PIECE\_WIDTH, PIECE\_WIDTH);

fillRect(&rPieceLast, PIECE\_WIDTH, 0, PIECE\_WIDTH, PIECE\_WIDTH);

fillRect(&rPieceTarget, PIECE\_WIDTH, PIECE\_WIDTH, PIECE\_WIDTH, PIECE\_WIDTH);

fillRect(&rButtonRegret[0], 0, 0, BUTTON\_WIDTH, BUTTON\_HEIGHT);;

fillRect(&rButtonRegret[1], BUTTON\_WIDTH, 0, BUTTON\_WIDTH, BUTTON\_HEIGHT);;

fillRect(&rButtonRegret[2], BUTTON\_WIDTH \* 2, 0, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonQuit[0], 0, BUTTON\_HEIGHT, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonQuit[1], BUTTON\_WIDTH, BUTTON\_HEIGHT, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonQuit[2], BUTTON\_WIDTH \* 2, BUTTON\_HEIGHT, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonHuman[0], 0, BUTTON\_HEIGHT \* 2, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonHuman[1], BUTTON\_WIDTH, BUTTON\_HEIGHT \* 2, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonHuman[2], BUTTON\_WIDTH \* 2, BUTTON\_HEIGHT \* 2, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonComputer[0], 0, BUTTON\_HEIGHT \* 3, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonComputer[1], BUTTON\_WIDTH, BUTTON\_HEIGHT \* 3, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonComputer[2], BUTTON\_WIDTH \* 2, BUTTON\_HEIGHT \* 3, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonBlack, 0, BUTTON\_HEIGHT \* 4, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonWhite, 0, BUTTON\_HEIGHT \* 5, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonDraw, 0, BUTTON\_HEIGHT \* 6, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonLater[0], BUTTON\_WIDTH \* 2, BUTTON\_HEIGHT \* 4, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonLater[1], BUTTON\_WIDTH \* 2, BUTTON\_HEIGHT \* 5, BUTTON\_WIDTH, BUTTON\_HEIGHT);

fillRect(&rButtonLater[2], BUTTON\_WIDTH \* 2, BUTTON\_HEIGHT \* 6, BUTTON\_WIDTH, BUTTON\_HEIGHT);

//按钮初始化

prButton1 = &rButtonHuman[0];

prButton2 = &rButtonComputer[0];

prButton3 = &rButtonQuit[0];

//辅助步数元初始化

stepX[0] = stepY[0] = -1;

}

//获得棋盘确定位置确定方向的字符串

//参数:

//str:棋子转换成的字符串, 字符串长度为11, ‘0’表示无棋, ‘1’表示己方棋子, ‘2’表示对方棋子

//(posX, posY):着棋位置

//dir:判断的方向

//pBlockL, pBlockR:左边和右边首个对方棋子的位置, 边界处也默认为对方棋子

//tempX, tempY:当前计算位置

void getPieceStr(char \*str, int posX, int posY, int dir, int\* pBlockL, int\* pBlockR) {

int j, tempX, tempY;

tempX = posX;

tempY = posY;

\*pBlockL = -1;

\*pBlockR = TOTAL\_JUDGE;

for (j = 0; j <= FIVE; ++j) {

if (out(tempX, tempY))

str[FIVE - j] = '2';

else

str[FIVE - j] = getPieceChar(map[tempX][tempY], map[posX][posY]);

if (str[FIVE - j] == '2') {

\*pBlockL = FIVE - j;

break;

}

tempX -= moveX[dir];

tempY -= moveY[dir];

}

tempX = posX;

tempY = posY;

for (j = 0; j <= FIVE; ++j) {

if (out(tempX, tempY))

str[FIVE + j] = '2';

else

str[FIVE + j] = getPieceChar(map[tempX][tempY], map[posX][posY]);

if (str[FIVE + j] == '2') {

\*pBlockR = FIVE + j;

break;

}

tempX += moveX[dir];

tempY += moveY[dir];

}

}

//估价函数

//返回:估价值

//参数:

//(posX, posY):估价的位置

//scSum:估价总和, scTemp:当前方向估价值

//scMax:当前方向最大估价值, st:判断的棋型

//str:当前方向转换成的字符串, 字符串长度为11, ‘0’表示无棋, ‘1’表示己方棋子, ‘2’表示对方棋子

//blockL, blockR:左边和右边首个对方棋子的位置, 边界处也默认为对方棋子

//通过4个方向的分值之和来判断该步着棋的总分值

//杀棋会有额外奖励分值

int judge(int posX, int posY) {

int i, j, scSum = 0, scTemp, scMax, st;

int blockL, blockR;

char str[TOTAL\_JUDGE + 1] = {"00000000000"};

for (i = 0; i < TOTAL\_DIRECTION; ++i) {

getPieceStr(str, posX, posY, i, &blockL, &blockR);

if (blockR - blockL <= FIVE)

continue;//此处不能形成五子连珠, 估价为0

j = (blockL == -1 ? 0 : blockL);

scMax = -INFI;

while (j < FIVE) {

while (str[++j] != '1');

for (st = 0; st < TOTAL\_STYLE; ++st)

if (strncmp(str + j, style[st], styleLen[st]) == 0)

break;

scTemp = 0;

if (st != TOTAL\_STYLE)

scTemp = score[st];

if (st != TOTAL\_STYLE && st > 1)

{

if (j - 1 == blockL && j + styleLen[st] == blockR)

scTemp /= TEN;//有2个对方棋子阻挡, 减少得分

if (j - 1 == blockL || j + styleLen[st] == blockR)

scTemp /= TEN;//有1个对方棋子阻挡, 减少得分

}

//满足某阵型, 计算阵型得分

if (scTemp > scMax)

scMax = scTemp;//取最大阵型得分

}

if (scSum >= SCORE\_DEFEND && scMax >= SCORE\_DEFEND)

scSum += scMax \* TEN;//杀棋额外奖励分值

else

scSum += scMax;

}

return scSum;

}

//判断函数

//判断着该步棋能形成的最多连子数

//返回:最多连子数

//参数:

//(posX, posY):判断位置

//st:判断的棋型

//winCase:用于返回

//blockL, blockR:左边和右边首个对方棋子的位置, 边界处也默认为对方棋子

//str:当前方向转换成的字符串, 字符串长度为11, ‘0’表示无棋, ‘1’表示己方棋子, ‘2’表示对方棋子

//如果为孤子,默认为2连,这样做不影响基于该判断的估价

int judgeWin(int posX, int posY) {

int i, j, st;

int winCase = 2;

int blockL, blockR;

char str[TOTAL\_JUDGE + 1] = {"00000000000"};

for (i = 0; i < TOTAL\_DIRECTION; ++i) {

getPieceStr(str, posX, posY, i, &blockL, &blockR);

if (blockR - blockL <= FIVE)

continue;

j = (blockL == -1 ? 0 : blockL);

while (j < FIVE) {

while (str[++j] != '1');

for (st = 1; st <= 8; ++st)

if (strncmp(str + j, style[st], styleLen[st]) == 0)

break;

if (st <= 8) {

if (styleWin[st] == 5)

winCase = 5;

if (styleWin[st] == 4 && styleWin[st] > winCase)

winCase = 4;

if (styleWin[st] == 3 && j - 1 > blockL && j + styleLen[st] < blockR && styleWin[st] > winCase)

winCase = 3;

}

}

}

return winCase;

}

//AI计算函数

//最优分值 = max{己方落子分值 – max{对方落子分值}}

//若为己方必杀, 则不计算对方落子分值

//若为对方必杀, 则不计算己方落子分值

//参数: (pPosX, pPosY):AI计算最优着棋位置, x:AI棋子颜色, count:当前步数

//max:AI最优分值, maxEnemy:玩家下一步最优分值, maxEnemyPos:玩家下一步最优位置数

//sc:AI当前分值, scEnemy: 玩家下一步当前分值

//nowWin: AI当前连子数, nextWin: 玩家下一步当前连子数

//maxNowWin: AI最大连子数, maxNextWin: 玩家下一步最大连子数

//mustKill: AI是否必杀

//(ti, tj):AI当前位置, (ni, nj):玩家下一步当前位置

//(bx, by):AI最优位置, (bbx, bby):玩家下一步最优位置

//SPECIAL:为了不让玩家通过计算时间来发现杀棋, 故没有为杀棋增加break

void compute(int \*pPosX, int \*pPosY, enum PIECE x, int count) {

int max = -INFI, maxEnemy = -INFI, maxEnemyPos = 0, maxTemp = 0;

int sc = 1, scEnemy = 0, nowWin = 0, maxNowWin = 0, nextWin = 0, maxNextWin = 0;

int mustKill = 0;

int ti, tj, ni, nj, bx = 0, by = 0, bbx, bby;

for (ti = 0; ti < TOTAL\_LINE; ++ti)

for (tj = 0; tj < TOTAL\_LINE; ++tj)

if (map[ti][tj] == NOTHING) {

map[ti][tj] = x;

sc = judge(ti, tj);

nowWin = judgeWin(ti, tj);

maxEnemy = -INFI;

maxEnemyPos = 0;

maxNextWin = 0;

for (ni = 0; ni < TOTAL\_LINE; ++ni) {

for (nj = 0; nj < TOTAL\_LINE; ++nj) {

if (map[ni][nj] == NOTHING) {

map[ni][nj] = getReversePiece(x);

nextWin = judgeWin(ni, nj);

scEnemy = judge(ni, nj);

if (maxNextWin < nextWin)

maxNextWin = nextWin;

if (scEnemy >= SCORE\_MUST\_DEFEND && (nowWin < nextWin))

sc = 0;//杀棋, 必赢

map[ni][nj] = NOTHING;

if (scEnemy > maxEnemy) {

if (maxEnemy < SCORE\_MUST\_DEFEND)

maxEnemyPos = 1;

else

++maxEnemyPos;

maxEnemy = scEnemy;

bbx = ni;

bby = nj;

}

else

if (scEnemy == maxEnemy || scEnemy >= SCORE\_MUST\_DEFEND)

++maxEnemyPos;//分数相等时重复计算

}

}

}

if (sc >= SCORE\_MUST\_DEFEND && nowWin >= maxNextWin && nowWin >= maxNowWin) {

if (sc > max) {

max = sc;

bx = ti;

by = tj;

}

mustKill = 1;

maxNowWin = nowWin;

printf("MUST KILL(%d, %d) now = %d next = %d\n", ti, tj, nowWin, maxNextWin);

**//[调试信息] 必杀**

//SPECIAL:为了不让玩家通过计算时间来发现杀棋, 故没有为杀棋增加break

}//杀棋, 必赢

map[ti][tj] = NOTHING;

if (!mustKill) {

maxEnemy \*= mulMul(maxEnemyPos);

maxTemp = ((x == BLACK ? timeMul(count) : 1.90 - timeMul(count)) \* distPlus(count, ti, tj)) \* sc - maxEnemy;

if (maxTemp > max || maxTemp == max && rand() % 100 < 25) {

max = maxTemp;

bx = ti;

by = tj;

}

}

printf("(%d, %d), (%d, %d) = %d , %d = %d\n", ti, tj, bbx, bby, sc, maxEnemy, maxTemp);

**//[调试信息] 各类位置的判断**

}

printf("Choose(%d, %d) = %d\n", bx, by, max);

**//[调试信息] AI走子**

\*pPosX = bx;

\*pPosY = by;

}

//游戏进程

//参数：

//flag:1(双人对战), 0(人机对战)

//gameOver:是否游戏结束

//result:是否某方获胜

//firstCom:firstMake:用于判断玩家是否在人机对战中选择后手

//count:步数

//(posX, posY):当前着棋位置

//x:着棋棋子

//gameEvent: SDL游戏事件变量

void game(int flag) {

int gameOver = 0;

int result = 0;

int firstCom = 0;

int firstMake = flag;

int count = 0;

int posX = -1, posY = -1;

enum PIECE x = BLACK;

union SDL\_Event gameEvent;

while (!gameOver && !systemOver) {

flip(count);

while (SDL\_PollEvent(&gameEvent)) {

systemOver = (gameEvent.type == SDL\_QUIT);

if (validMove(&gameEvent)) {

switch (gameEvent.type) {

case SDL\_MOUSEMOTION:

switch (getButtonMove(gameEvent.motion.x, gameEvent.motion.y)) {

case 1: if (!firstMake) prButton1 = &rButtonLater[1]; break;

case 2: if (!result) prButton2 = &rButtonRegret[1]; break;

case 3: prButton3 = &rButtonQuit[1]; break;

default:

if (!result)

prButton2 = &rButtonRegret[0];

if (!flag)

prButton1 = firstMake ? &rButtonComputer[2] : &rButtonLater[0];

prButton3 = &rButtonQuit[0];

} break;//停留按钮高亮

case SDL\_MOUSEBUTTONDOWN:

switch (getButtonMove(gameEvent.motion.x, gameEvent.motion.y)) {

case 1: if (!firstMake) prButton1 = &rButtonLater[2]; break;

case 2: if (!result) prButton2 = &rButtonRegret[2]; break;

case 3: prButton3 = &rButtonQuit[2]; break;

} break;//按下按钮高亮

case SDL\_MOUSEBUTTONUP:

switch (getButtonMove(gameEvent.motion.x, gameEvent.motion.y)) {

case 1: if (!firstMake) {

firstCom = 1;

firstMake = 1;

prButton1 = &rButtonComputer[2];

++count;

stepX[count] = TOTAL\_LINE >> 1;

stepY[count] = TOTAL\_LINE >> 1;

map[stepX[count]][stepY[count]] = x;

x = getReversePiece(x);

}

case 2: //悔棋

if (flag) {

if (count >= 1) {

fprintf(fp, "Regret STEP%d\n", count);

**//[调试信息] 悔棋**

map[stepX[count]][stepY[count]] = NOTHING;

stepX[count] = stepY[count] = -1;

--count;

x = getReversePiece(x);

}

prButton1 = &rButtonHuman[2];

}

else {

if (!result && count > 1) {

fprintf(fp, "Regret step%d\n", count);

**//[调试信息] 悔棋**

map[stepX[count]][stepY[count]] = NOTHING;

stepX[count] = stepY[count] = -1;

--count;

map[stepX[count]][stepY[count]] = NOTHING;

stepX[count] = stepY[count] = -1;

--count;

}

prButton1 = &rButtonComputer[2];

}

break;

case 3: //退出

gameOver = 1;

break;

}

break;

}

flip(count);

if (!result && validPieceMove(gameEvent.motion.x) && validPieceMove(gameEvent.motion.y)) {

stepX[count + 1] = posX = getPieceMove(gameEvent.motion.x);

stepY[count + 1] = posY = getPieceMove(gameEvent.motion.y);

if (map[posX][posY] == NOTHING && gameEvent.type == SDL\_MOUSEBUTTONUP) {

//双人对战

if (flag) {

++count;

stepX[count] = posX;

stepY[count] = posY;

map[posX][posY] = x;

x = getReversePiece(x);

printf("STEP %d : (%d, %d)\n", count, posX, posY);

printf("JUDGE %d\n", judgeWin(posX, posY));

**//[调试信息] 着棋**

if (judgeWin(posX, posY) >= 5) {

result = 1;

prButton2 = (x == WHITE ? &rButtonBlack : &rButtonWhite);

break;

}

}

//人机对战

else {

firstMake = 1;

prButton1 = &rButtonComputer[2];

++count;

stepX[count] = posX;

stepY[count] = posY;

fprintf(fp, "HUM: (%d, %d)\n", posX, posY);

**//[调试信息] 着棋**

map[posX][posY] = x;

x = getReversePiece(x);

if (judgeWin(posX, posY) >= 5) {

result = 1;

prButton2 = (x == WHITE ? &rButtonBlack : &rButtonWhite);

break;

}

if (result) break;

flip(count);

++count;

if (count < TOTAL\_PIECE) {

compute(&posX, &posY, x, count);

fprintf(fp, "COM: (%d, %d)\n", posX, posY);

**//[调试信息] 着棋**

stepX[count] = posX;

stepY[count] = posY;

map[posX][posY] = x;

x = getReversePiece(x);

if (judgeWin(posX, posY) >= 5) {

result = 1;

prButton2 = (x == WHITE ? &rButtonBlack : &rButtonWhite);

}

}//白棋胜

if (result) break;

}

//统计结果

}

}

}

}

if (count == TOTAL\_PIECE) {

result = 1;

prButton2 = &rButtonDraw;

} //平局

SDL\_Delay(TIME\_DELAY);

}

}

//主函数

//参数:

//gameEvent:SDL游戏事件变量

int main(int argc, char \*argv[]) {

union SDL\_Event gameEvent;

init();

fresh();

while (!systemOver) {

flip(0);

while (SDL\_PollEvent(&gameEvent)) {

systemOver = (gameEvent.type == SDL\_QUIT);

switch(gameEvent.type) {

case SDL\_MOUSEMOTION: //移动按钮

switch (getButtonMove(gameEvent.motion.x, gameEvent.motion.y)) {

case 1:

prButton1 = &rButtonHuman[1];

break;

case 2:

prButton2 = &rButtonComputer[1];

break;

case 3:

prButton3 = &rButtonQuit[1];

break;

default:

prButton1 = &rButtonHuman[0];

prButton2 = &rButtonComputer[0];

prButton3 = &rButtonQuit[0];

} break;

case SDL\_MOUSEBUTTONDOWN: //按下按钮

switch (getButtonMove(gameEvent.motion.x, gameEvent.motion.y)) {

case 1: prButton1 = &rButtonHuman[2]; break;

case 2: prButton2 = &rButtonComputer[2]; break;

case 3: prButton3 = &rButtonQuit[2]; break;

} break;

case SDL\_MOUSEBUTTONUP://单击按钮

switch (getButtonMove(gameEvent.motion.x, gameEvent.motion.y)) {

case 1:

prButton1 = &rButtonHuman[2];

prButton2 = &rButtonRegret[0];

prButton3 = &rButtonQuit[0];

printf("Human VS Human\n");

fprintf(fp, "Human VS Human\n");

**//[调试信息] 选择类别**

game(1);

prButton1 = &rButtonHuman[0];

prButton2 = &rButtonComputer[0];

prButton3 = &rButtonQuit[0];

fresh();

break;

case 2:

prButton1 = &rButtonLater[0];

prButton2 = &rButtonRegret[0];

prButton3 = &rButtonQuit[0];

printf("Human VS Computer\n");

fprintf(fp, "Human VS Computer\n");

**//[调试信息] 选择类别**

game(0);

prButton1 = &rButtonHuman[0];

prButton2 = &rButtonComputer[0];

prButton3 = &rButtonQuit[0];

fresh();

break;

case 3:

printf("System Quit\n");

**//[调试信息] 退出系统**

systemOver = 1;

break;

}

break;

}

}

SDL\_Delay(TIME\_DELAY);

//刷新屏幕延时

}

//关闭文件/退出

fclose(fp);

SDL\_Quit();

return 0;

}